# Assignment: Homework 5

## What to Hand In

1. Codes that can reproduce all your results. It is your responsibility to make all your codes readable. Detailed comments and meaningful variable names are highly recommended.
2. Your saved trained model and the script to run your model.
3. A report (PDF) that describes your attempts. You should include what you have done and discussions on cases when your model works and fails, along with possible explanations and potential improvements. You are free to cite published papers in your report, but it is prohibited to cite their code in your code.
4. Any other files if necessary.

## How to Hand It In

1. Put all your solutions **including a report** showing your answers and figures in one folder. Compress this folder and name it <firstname>_<lastname>_HW5.zip. For example, "Zhiyao_Duan_HW5.zip".
2. Submit to the appropriate entry on Blackboard.

## When to Hand It In

It is due at 11:59 PM on the date specified on the course calendar. **Late assignments will receive a 20% deduction of the full grade each day.**

## Problems (10 points in total)

This homework will guide you through the process of applying neural networks to an important audio processing task: separating singing voice from a song. The basic approach is as follows:

1. Transform the signal into its STFT domain.
2. For the magnitude spectrum $\boldsymbol{s_t}$ of the t-th frame, we predict a mask $\boldsymbol{m_t}$ and then calculate the spectrum $\boldsymbol{v_t}$ of the singing voice as
$$\boldsymbol{v_t} = \boldsymbol{m_t} \odot \boldsymbol{s_t},$$
where $\odot$ represents element-wise multiplication.
3. Reconstruct the estimated singing voice using our estimated magnitude spectrum and the original mixture's phase with inverse STFT and overlap add or the Griffin-Lim algorithm.

You are allowed to make modifications to the architecture or parameters shown in the following sections for a reason.

1. (2 points) Dataset. Download the dataset (https://sigsep.github.io/datasets/dsd100.html). The first step is to understand each function in the Dataset class in "Data.py" that returns linear spectrograms of mixture and voices by the index. Then use it in Dataloader class to collect them into batches.
   **Note 1:** To reduce the size of the dataset you can delete all tracks from the sources folders except vocal.wav.
   **Note 2:** To have more training data, you can copy 20 files from the test folder to the train folder and split the rest 30 files into 15 songs for validation and 15 songs for test. Use validation set for the model tuning (such as learning rate, batch size, number of frames, etc.).

**Note 3:** It is better to take a random chunk (with a fixed size) of the spectrogram of each song instead of the entire spectrogram when you form a batch to train the model (see Problem 3). It can be done in the Dataset class only for feedforward model (Problem 3), and in the training loop for both models (see note for Problem 4).
**Note 4:** To avoid zeros in the data, you can use method clamp (1e-5).
**Note 5:** If you want to reduce the amount of data, you can use window size 1024 instead of 4096 samples, then your spectrogram will have 513 bins instead of 2049. The other option is to downsample the audio to 32kHz.

2. (0.5 points) Data augmentation. You can add a silence chunk of an arbitrary length smaller than hop_size in the beginning of the audio in the dataset class.
**Note:** you can add it after model implementation (Tasks 3 and 4) and check out if it improves the performance.

3. (3.5 points) Feedforward model + training code ("Model.py"). The first model you need to build is a single-frame encoder-decoder architecture for estimating the vocal spectral mask from the mixture (Figure 1). Train your model. Save your best performing model (lowest validation loss) as savedModel_feedforward_best.pt. Fill in blanks in "separate_feedforward.py" to run your model as a command line tool on a specific audio file. The output of the encoder can be used as the input for the recurrent layer in the next section.

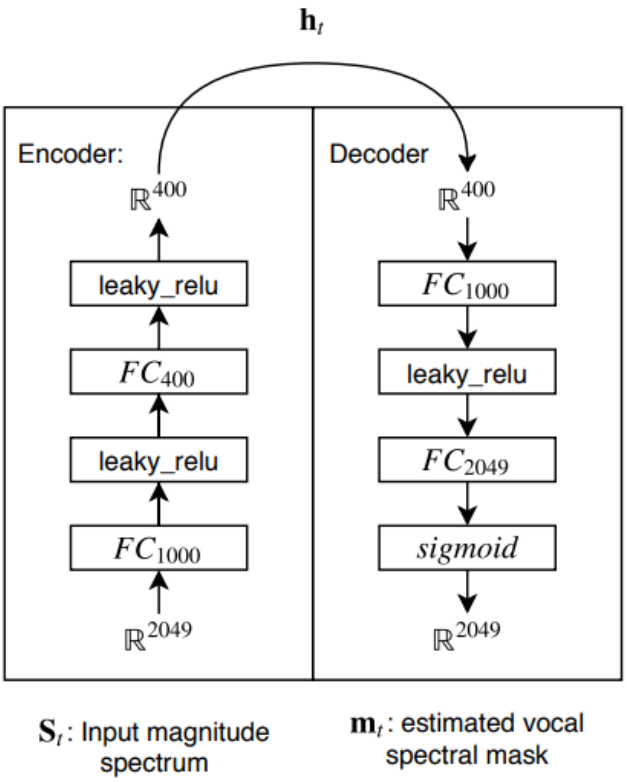



Figure 1: The encoder-decoder architecture for the single frame model. FC means the fully-connected layer (nn.Linear in Pytorch ). Here we use FFT size 4096 and hop size 2048.

The default loss function for this assignment is the generalized Kullback-Leibler divergence. You have seen this divergence in your previous assignment:

$$\min E[D(\boldsymbol{u_t} \parallel \boldsymbol{v_t})],$$

where $\boldsymbol{v_t}$ is the estimated spectrum for the frame $\boldsymbol{s_t}$, and $\boldsymbol{u_t}$ is the target spectrum of singing voice. Generalized Kullback-Leibler divergence is given by

$$D(X \parallel Y) = \sum_{i,j} x_{ij}\left(\log\left(x_{ij}+\varepsilon\right) - \log\left(y_{ij}+\varepsilon\right)\right) - x_{ij} + y_{ij},$$

where $\varepsilon$ is a small number added for numerical stability. Matrix notation is used here because you will deal with multiple instances batched together. The final value of the divergence should be averaged over batch size and the sequence length to make the loss for different batch sizes and sequence lengths comparable. Note that swapping X and Y in the KL divergence gives a different objective. You can try this if you are interested.

You can monitor the training progress by tracking the moving average on the noisy loss value:

$$loss_{smoothed} = \alpha loss_{smoothed} + (1-\alpha) loss_{current},$$

where α ∈ (0, 1), usually very close to 1 (e.g., 0.99). The loss curve helps you decide when to stop training. You can use matplotlib to do some basic plotting and save the figure to the disk periodically.

**Note 1:** You may use built-in MSE loss function or KL divergence before the generalized KL divergence.

**Note 2:** If you use TensorBoard, you can use the built-in moving average plot of loss function.

4. (2 points) RNN model. The second model incorporates temporal information. The architecture is shown in Figure 2.
   **Note 1:** To reduce the computational graph you can do the backward step every N frames (e.g., 5, 20, 30) initializing hidden (h_0) and cell (c_0) tensors of the LSTM layer with h_N and c_N, the outputs from the previous run of the LSTM layer. It can be done in an additional "for" loop over the frames of the song, i.e., initialize (h_0, c_0) by zeros before the "for" loop, take the first N frames of the spectrogram, do optimizer.step(); for the next N frames use (h_0, c_0) = (h.detach(), c.detach()), where (h, c) is the output of the LSTM layer after optimizer.step(); and continue till the end of the frames. For the next batch, initialize (h_0, c_0) with zeros again. After training, when you do inference with the trained model, i.e., calling model.eval(), you do not need this additional loop; Instead, just pass the entire song with zero initial states. (h_0 here is a parameter of LSTM and is different from h_0 in Fig. 2.)
   **Note 2:** it is recommended to directly reuse your trained feedforward model to initialize some of the weights of this model to speed up training.

5. (1 point) Inference. You can use inverse STFT function with original phase or the Griffin-Lim function with the same parameters that you used for the spectrogram. You can do the training and evaluation in Jupyter or Python. But we ask you to write a script that can be run on any .wav file. Use your saved model to fill in the separate_feedforward.py and separate_rnn.py templates.

6. (1 point) Accuracy estimation. A simple way to measure the performance of your model is to calculate the signal-to-distortion ratio (SDR, in decibels) using the ground truth and your estimated output on the test portion of the dataset. You can calculate SDR with BSS_EVAL 3.0 toolkit you used in the previous assignment (in this single source case, SDR still works). Please include evaluation results by reporting the average SDR value on the whole test set, to compare all your models.

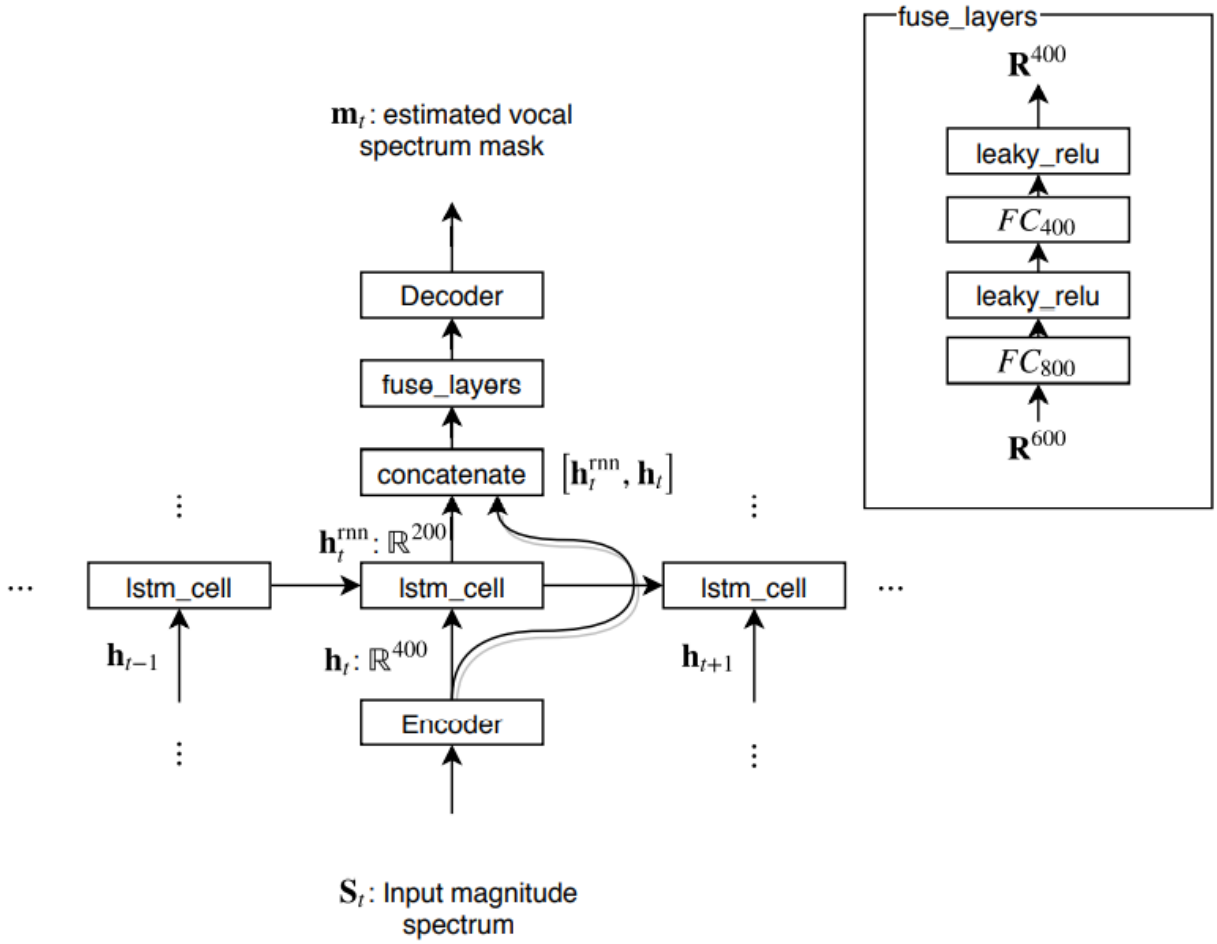



Figure 2: An LSTM layer takes the output h_t of the encoder and outputs h_t^rnn in order to summarize information across frames. h_t and h_t^rnn are then concatenated and fed into the fuse_layers. The output of the fuse_layers are then fed into the decoder for obtaining the estimated mask.